Master Thesis

A multi-resolution particle method with semi-implicit discretisation for membrane dynamics simulations

Author(s):
Mitrović, Milan

Publication Date:
2011

Permanent Link:
https://doi.org/10.3929/ethz-a-006686905

Rights / License:
In Copyright - Non-Commercial Use Permitted

This page was generated automatically upon download from the ETH Zurich Research Collection. For more information please consult the Terms of use.
A multi-resolution particle method with semi-implicit discretisation for membrane dynamics simulations

Master Thesis
Milan Mitrović
October 10, 2011

Supervisors: Dr. Sylvain Reboux, Prof. Dr. Ivo F. Sbalzarini
Department of Computer Science, ETH Zürich
Abstract

Understanding the way cells control the shape of their membrane is a fascinating problem for which the principles of operation are currently being uncovered. As the relevant variables are not controllable or observable in wet experiments there is a need for efficient simulation algorithms that can be used to examine various hypothesis on the detailed mechanics of these processes. The physical model we choose for this problem is the minimisation of the bending energy of an elastic membrane with spatially varying spontaneous curvature and bending rigidity. The membrane surface is represented in implicit form as a zero level of a phase field function that is simulated using a pure particle simulation method.

In this formulation the governing equation is a non-linear sixth-order partial differential condition. The stability condition of fully explicit numerical schemes is in that case severely limiting. The goal of this master thesis is to combine an implicit time stepping scheme with a time adaptation method that constrains the local temporal error, and a spatial adaptation method that can take advantage of the thin region of interest in the simulation domain and track its position through time. The method is verified and shown to have good performance on the heat equation. The application of the method to the phase field equation is shown to have some performance limitations but also leaves promising possibilities for further improvements.
Acknowledgements

This thesis would not have been possible without constant contribution and support from my advisor, Dr. Sylvain Reboux. Thank you for being a great mentor and friend. To Prof. Ivo Sbalzarini: thank you for giving me this opportunity and for always finding time to answer my questions. No matter the topic I am always surprised by your keen intuition. I am very grateful that I have the opportunity to work with all the people in the MOSAIC group. Your support and encouragement make me feel at home. I would also like to thank my family for all their love and support and for making this possible in the first place. To Alex and Vila, thank you for being so strong.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Organisation of this report</td>
<td>3</td>
</tr>
<tr>
<td>2 Theoretical Background</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Spatio-temporal Simulations using Particles</td>
<td>5</td>
</tr>
<tr>
<td>2.1.1 Function Approximation</td>
<td>6</td>
</tr>
<tr>
<td>2.1.2 Operator Approximation</td>
<td>7</td>
</tr>
<tr>
<td>2.1.3 Algorithm</td>
<td>9</td>
</tr>
<tr>
<td>3 Implicit Solver</td>
<td>11</td>
</tr>
<tr>
<td>3.1 Heat Equation</td>
<td>11</td>
</tr>
<tr>
<td>3.2 Explicit Time Stepping</td>
<td>12</td>
</tr>
<tr>
<td>3.3 Implicit Time Stepping</td>
<td>14</td>
</tr>
<tr>
<td>4 Adaptive resolution</td>
<td>19</td>
</tr>
<tr>
<td>4.1 Adaptive Time Stepping</td>
<td>19</td>
</tr>
<tr>
<td>4.2 Self-Organizing Lagrangian Particles</td>
<td>22</td>
</tr>
<tr>
<td>4.3 Parallel Performance</td>
<td>25</td>
</tr>
<tr>
<td>5 Membrane Dynamics</td>
<td>27</td>
</tr>
<tr>
<td>5.1 Theoretical Background</td>
<td>27</td>
</tr>
<tr>
<td>5.1.1 Differential Geometry of Surfaces</td>
<td>27</td>
</tr>
<tr>
<td>5.1.2 Physics of Cellular Membranes</td>
<td>28</td>
</tr>
<tr>
<td>5.2 Phase Field Method</td>
<td>29</td>
</tr>
<tr>
<td>5.3 Implementation</td>
<td>30</td>
</tr>
<tr>
<td>5.3.1 Jacobian Calculation</td>
<td>32</td>
</tr>
<tr>
<td>5.4 Simulation</td>
<td>34</td>
</tr>
<tr>
<td>5.4.1 Implicit Solver</td>
<td>36</td>
</tr>
<tr>
<td>5.4.2 Adaptive Time</td>
<td>38</td>
</tr>
</tbody>
</table>
6 Conclusions

A PPM–PETSc utilities

List of Figures

Bibliography
Chapter 1

Introduction

The goal of this master thesis is to develop an efficient particle simulation of cellular membrane dynamics using the Parallel Particle Mesh [13] library (PPM). The physics of cellular membranes is a very challenging subject that is not yet completely understood. Interest in this problem has been very high in recent years as advances in technology created new ways of studying their properties, both experimentally and computationally. On the microscopic scale, molecular dynamics simulations enabled testing the different models of microscopic behavior, from which the macroscopic properties can be calculated. On the macroscopic scale cells are modeled as closed, elastic surfaces deforming under different forces. On this scale the mechanisms by which cells control their shape can be studied by coupling simulations of the physics of the membrane with simulations of cellular regulatory systems.

The most common ways of modeling surfaces are direct methods, front tracking, volume of fluid (VOF) and level set methods. Direct methods use a discretisation mesh on the surface itself, which is then treated as a boundary between the internal and external domain. Although conceptually simple, these methods require sophisticated extensions in order to accurately track the surface in case of large deformations or topological changes. It is also very difficult to couple the surface forces with the fields of other phenomena in the environment of the membrane.

Methods that use an implicit description of the surface, such as level sets, VOF and front tracking, can be more easily coupled with other fields, and are more suitable for treatment of large deformations and topological changes. Phase field methods are a subclass of level set methods in which the surface motion is due to physical energy dissipation for some free energy functional (Du et. al. [3]). The functional is usually derived from a specific model of the bending energy of the surface (e.g. Willmore or Canham-Helfrich).

In front tracking methods, the surrounding continuum medium is discre-
1. Introduction

tised on a mesh and the surface is typically described by its intersection with the mesh and can be seen as a line of discontinuity for the data represented on the mesh. The surface can also be discretised by a separate, unstructured grid that moves through the mesh [16].

In VOF methods the domain is partitioned into small volume elements and the interface is represented implicitly by a color function that takes values 1 and 0 inside and outside of a closed surface. Only the integral value of the color function in each volume element is stored and evolved through a time stepping scheme [7]. VOF methods have the advantage of conserving the mass of the system. The main difficulty, however, lies in maintaining sufficient accuracy to prevent an excessive smearing of the interface and in computing mechanical properties such as surface tension.

The level-set method is reviewed by Osher and Fedkiw [9] and by Sethian [15]. Level-set methods by themselves do not incorporate any physics or mechanics and, as opposed to VOF methods, they do not preserve volume. One of their main advantages is that, in theory, the position of the interface can be tracked with sub-pixel resolution and with arbitrarily high order of accuracy.

Many approaches to solving phase field equations for membrane dynamics have been examined in recent years. Du et. al. developed a method for simulation of axisymmetric membranes and thus reduce the problem to two dimensions [5]. The constraints are enforced using a penalty method and the evolution equation is discretised on a regular mesh and solved with finite differences and an implicit time stepping scheme. Du and Zhang used an adaptive finite element method for elastic energy minimization in [6]. A similar method with a penalty formulation of the problem is developed by Lowengrub et. al. to study budding and fission in 2D membrane simulations for the case of two phases in the membrane surface with an added line tension energy [8]. In [4] Du et. al. study the deformation of a cellular membrane in an external flow field. Wang and Du also study the dynamics of multi-component vesicles that minimize bending and line tension energy under constant surface and volume constraints using a penalty formulation of the problem and a spectral method with implicit time stepping [17].

In this thesis I minimize the bending energy of the membrane using a phase field description of the interface following a model developed in the PhD thesis by Felix Campelo [2]. The phase field also evolves under a constant volume constraint that is accounted for in the formulation of the problem, and a constant surface constraint that is enforced using Lagrange multipliers. The resulting equation is a non-linear sixth order partial differential equation (PDE) that is very expensive to solve with a fully explicit method.

Since this problem is very expensive to solve with an explicit scheme, I implement an implicit time stepping scheme using the Portable, Extensible
1.1. Organisation of this report

Toolkit for Scientific Computation [1] (PETSc) for the non-linear solver and create an easy to use interface from PPM. For the time step size I implement an adaptive scheme based on a local temporal error approximation. The spatial discretisation is done using the SOP method developed in the MOSAIC group [10] which takes advantage of the thin region of interest of the phase field formulation.

1.1 Organisation of this report

In Chapter 2 I give a short overview of particle methods by summarizing their theoretical foundations and sketching the general algorithm. In Chapter 3 I present a simulation of the heat equation using a standard explicit scheme and show the results that serve as reference for later comparisons. Then I show the implementation of an implicit time stepping scheme based on PETSc and present the performance results.

In Chapter 4 I add a time adaptivity scheme to the implicit solver developed in the previous chapter. The method is then completed with spatial adaptation as implemented in SOP to give a fully adaptive simulation method. The performance of the simulation method is also tested in parallel.

Finally, in Chapter 5 the method is applied to the equation of cellular membrane dynamics where its performance and drawbacks are determined.
In this chapter I give a short overview of the theory of particle simulation methods. Most of the content in this chapter is derived from the course Spatiotemporal Modelling and Simulation at ETH which is taught by Ivo Sbalzarini. For further reference on any of the topics discussed here please see the lecture notes provided with the course [12].

2.1 Spatio-temporal Simulations using Particles

Numerically solving a Partial Differential Equation of a spatio-temporal problem requires the discretisation of the solution in space and time as the infinitely many states of the continuous function cannot be stored in a computer with finite memory. One way to do this is to store information of the function value only on a finite set of points scattered in space. There are two major ways to choose these discretisation points: placing them uniformly on a regular lattice or placing them irregularly so that they vary in density in different regions of space. Particle methods give the flexibility of placing the discretisation points in both ways and also allow them to move during the course of the simulation.

In spatio-temporal problems we are usually interested in calculating the evolution of some field in space given a known initial condition. As with space, the solution cannot be calculated at every point in time. Instead we repeatedly calculate where the field will move in one time step, by using the information from the previous states of the field along with the chosen size of the step. As we are usually only interested in the state of the field at some final time point we need to store only the last few states of the field that are necessary to calculate the next step. What we always do have to take care of is that the time steps are small enough to resolve the dynamics of the phenomena of interest because otherwise the numerical solution will be very far from reality.
Finally, we will also need a way to compute approximations of differential operators on fields stored in a discretised representation. The flexibility of placing particles freely in space comes at a cost of slightly more complex operator approximation schemes.

2.1.1 Function Approximation

To approximate a continuous function \( u(x) : \mathbb{R}^d \to \mathbb{R} \) using a finite number of particles we need to have a way of evaluating the function at off-particle positions. Let’s start from the Dirac-delta identity

\[
    u(x) = \int u(y) \delta(x - y) dy. \tag{2.1}
\]

This integral states that the function is equal to its convolution with the Dirac delta. It can already be discretised by turning it into a sum over the selected particle positions instead of the whole domain of \( y \) but this would give a point wise sampling of the function and the value would be unknown at off-particle positions.

Mollification

To get a smooth (mollified) approximation we will replace the Dirac-delta with a smooth kernel function \( \zeta \) with local support.

The kernel function is further restricted to a finite width \( \epsilon \) by setting \( \zeta_{\epsilon} = \epsilon^{-d} \zeta(\frac{x}{\epsilon}) \) so that

\[
    \lim_{\epsilon \to 0} \zeta_{\epsilon} = \delta, \tag{2.2}
\]

which requires that \( \int \zeta dx = 1 \). The function \( \zeta \) is called the mollification kernel of characteristic width \( \epsilon \). The mollified function approximation then becomes

\[
    u_{\epsilon}(x) = \int u(y) \zeta_{\epsilon}(x - y) dy. \tag{2.3}
\]

Mollification introduces an error that depends on the kernel \( \zeta \) and can be shown to be

\[
    u_{\epsilon}(x) = u(x) + O(\epsilon^r), \tag{2.4}
\]
where \( r \) is the order of the highest conserved moment of \( \delta \)

\[
\int x^s \zeta(x) dx = \int x^s \delta(x) dx = \begin{cases} 
1 & s = 0 \\
0 & s \in [1, r - 1]
\end{cases}.
\] (2.5)

Equation (2.5) gives us a set of conditions (called moment conditions) that a good mollification kernel has to satisfy. Therefore these conditions can be applied directly to engineer mollification kernels of any desired level of accuracy.

**Discretisation**

Now that we have a smooth kernel we can discretise the integral of the mollified function approximation (2.3) over the finite set of particles

\[
u^h_\varepsilon(x) = \sum_{p=1}^{N} \omega^h_\varepsilon(x - x^h_p),
\] (2.6)

where \( x^h_p \) and \( \omega^h_p \) are the positions and strengths of particles. The total error now has two components: the mollification error and the discretisation (quadrature) error

\[
u^h_\varepsilon(x) = u(x) + O(\varepsilon^r) + O\left(\frac{h}{\varepsilon}\right)^s,
\] (2.7)

where \( s \) is the number of continuous derivatives of \( \zeta \) and \( h \) is the inter-particle distance. From this expression we see that

\[
\frac{h}{\varepsilon} < 1
\] (2.8)

in order for the error to be bounded. Equation (2.8) is called the overlap condition because it requires that the width of the mollification kernel is greater than the inter-particle spacing. This also intuitively makes sense as the function value would be unknown in regions where mollification kernels are not overlapping.

### 2.1.2 Operator Approximation

To derive an operator approximation lets start with the Taylor expansion of a function \( u \) around \( x \) in one dimension (the multidimensional case is analogous):

\[
u(y) = u(x) + (y - x) \frac{\partial u}{\partial x} + \frac{1}{2}(y - x)^2 \frac{\partial^2 u}{\partial x^2} + \frac{1}{6}(y - x)^3 \frac{\partial^3 u}{\partial x^3} + \cdots.
\] (2.9)
Now let's subtract $u(x)$ from the equation, multiply with an unknown kernel $\eta$, also normalized $\eta = \frac{1}{\varepsilon} \eta(\frac{x}{\varepsilon})$ to a finite width $\varepsilon$, and finally integrate over the entire domain $\Omega$:

$$\int_{\Omega} (u(y) - u(x)) \eta \varepsilon (y - x) dy = \int_{\Omega} (y - x) \frac{\partial u}{\partial x} \eta \varepsilon (y - x) dy$$

$$+ \frac{1}{2} \int_{\Omega} (y - x)^2 \frac{\partial^2 u}{\partial x^2} \eta \varepsilon (y - x) dy$$

$$+ \frac{1}{6} \int_{\Omega} (y - x)^3 \frac{\partial^3 u}{\partial x^3} \eta \varepsilon (y - x) dy$$

$$+ \cdots$$

This equation allows us to construct integral approximations of any derivative. By making a change of variables $z = (y - x)$ the equations on the right hand side become moments of the kernel $\eta$ multiplied with the respective derivative. By choosing the kernel $\eta$ so that all the moments save one (up to a certain order) are zero, we will get an equation that has only one derivative on the right hand side, and a left hand side that we can easily compute. Take note that the kernel $\eta$ derived using these moment conditions will in general be different from the mollification kernel $\zeta$.

Take for example the second derivative. Choosing an even $\eta$ makes all the odd integrals vanish. Further restricting it so that $\int z^2 \eta(z) dz = \frac{1}{2}$ and all the higher moments up to a certain order $r + 1$ are zero, we get the following:

$$\int_{\Omega} (u(y) - u(x)) \eta \varepsilon (y - x) dy = \frac{\partial^2 u}{\partial x^2} \varepsilon^2 + O(\varepsilon^{r+2}) \quad (2.10)$$

The factor $\varepsilon^2$ comes from the $\frac{1}{\varepsilon}$ in the normalization and the order of the computed second moment. Solving for the desired derivative we get:

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{\varepsilon^2} \int_{\Omega} (u(y) - u(x)) \eta \varepsilon (y - x) dy + O(\varepsilon^r) \quad (2.11)$$

Equation (2.11) is the integral operator approximation of the second derivative in 1D. The same technique can be used to derive approximations for any derivative.

To use equation (2.11) in particle simulations we can simply approximate this integral with a sum over the set of particles

$$\frac{\partial^2 u^h}{\partial x^2} (x_p^h) = \frac{1}{\varepsilon^2} \sum_{q=1}^{N} V_q (\omega_q^h - \omega_p^h) \eta \varepsilon (x_q^h - x_p^h) \quad (2.12)$$
where \( V_q \) is the particle volume required by the quadrature. This method is called Particle Strength Exchange (PSE) because it amounts to an exchange of strength between the interacting particles. The first term represents the difference in strength of two interacting particles and is multiplied by the kernel that combines this difference with inter-particle distance to get an exchange of strength.

### 2.1.3 Algorithm

Using the function and operator approximation schemes defined above a general sketch of the particle simulation method used in this thesis is shown in Algorithm 2.1.

**Algorithm 2.1** The general particle simulation algorithm.

1. Place the particles in a way that reflects the problem resolution requirements and determine the particle volumes \( V_p \)
2. Initialize the particle strengths \( \omega_p \) according to the initial condition of the problem and initialize the time variable
3. **while** \( t \leq t_{end} \) **do**
4. Compute the differential operators of the field using the particle strength exchange equation
5. Calculate the next state of the field using the calculated operator and the chosen time step
6. \( t = t + dt \)
7. \( u = u^{(t+1)} \)
8. **end while**

There are many choices for the time stepping scheme that can be used on line 5 of the Algorithm 2.1. They can be divided into two classes depending on how \( u_{t+1} \) is calculated. In explicit schemes the value of the next time step can be computed directly from the previous states of the simulation. In implicit schemes the next step can only be found as a solution of a system of equations that has to be solved at every time step. The effects of this choice and the advantages and drawbacks of each type of method will be examined in the rest of this thesis.
I will start this chapter by introducing the heat equation — a simple problem for which an analytical solution is known. Since both the analytical and numerical behaviors of the heat equation are well known, it allows for a systematic assessment of each of the new features added to PPM. Then I will describe the basic simulation method and verify its correctness using the heat equation.

The rest of the chapter will be focused on relaxing the time step limit by replacing explicit with an implicit time stepping scheme. This provides the basis for more advanced methods discussed later, but in turn requires numerically solving a system of equations at every time step. As the numerical solver increases the cost of every time step dramatically, the gain in time step size must be maximized in order to cover the added cost, while also making sure that the accuracy of the method is preserved.

### 3.1 Heat Equation

The isotropic, homogeneous heat equation is a second order, parabolic partial differential equation that describes the distribution of heat in space and time $u(\mathbf{x}, t)$

$$\frac{\partial u}{\partial t} = \alpha \Delta u, \quad (3.1)$$

where $\alpha$ is the thermal diffusivity. Throughout this and the next chapter I will test the methods by solving the heat equation numerically on the domain $[0, 1]^2$ with periodic boundaries for the initial condition

$$u(\mathbf{x}, 0) = \sin(3x) \sin(6y), \quad (3.2)$$
3. Implicit Solver

Figure 3.1: Convergence plot for the Forward Euler explicit time stepping scheme. The dashed line has slope 2 which is the theoretical prediction for convergence using a second order DC-PSE operator approximation. The result of the simulation is the black line and we can see that it matches the theoretical prediction very well.

where \( x = (x, y) \), until the time \( t_{\text{end}} = 0.1 \). The analytical solution for this initial condition is

\[
    u(x, t) = \sin(3x) \sin(6y) e^{-\alpha 45t}. \tag{3.3}
\]

Without loss of generality I show the results for \( \alpha = 1 \).

3.2 Explicit Time Stepping

The simplest explicit time stepping scheme is called Forward Euler and is based on the forward finite difference approximation of the first derivative

\[
    \frac{u(t+1) - u(t)}{dt} = F(u(t)). \tag{3.4}
\]

I use this scheme to implement the reference simulation of the heat equation using Algorithm 2.1 from the previous chapter (page 9) where the time step
3.3. Implicit Time Stepping

is computed using Forward Euler and the DC-PSE operator approximation [14] that is available with the PPM library:

\[ u_p^{(t+1)} = u_p^{(t)} + dt \alpha \sum_{q \in N(p)} \eta_{pq} (u_q^{(t)} - u_p^{(t)}). \] (3.5)

The particles are initialized on a grid of cell length \(dh\) and then randomly displaced by at most \(\frac{dh}{4}\) in both directions. The stability limit on time step size is of the order of \(dh^2\) where the exponent 2 comes from the order of the spatial derivatives in the right hand side of the heat equation.

The final relative error can be calculated using the analytical solution and should decrease with \(dh\) according to the chosen order of the operator approximation. In Figure 3.1 we can see the results when the order of the operator approximation was set to 2. The theoretical prediction (dashed line) matches the simulation results very well, which validates the implementation of the reference simulation and the DC-PSE operators of PPM.

Figure 3.2: Convergence of the implicit scheme as a function of \(dt\). The plot shows the final relative \(l_2\) error of the simulation.
3. Implicit Solver

3.3 Implicit Time Stepping

Implicit methods impose no stability restriction on the time step size but they require solving a system of equations at every time step. The wall-clock time cost of a time step is thus much greater, but it can be offset by choosing a large enough $dt$ i.e. such that the higher computational cost is covered. The numerical solution is accurate only if the real solution is sufficiently smooth during the chosen time step interval.

PPM does not include a solver for systems of non-linear equations, but there are a number of other libraries that do. After some consideration I chose to use PETSc as it includes the largest number of solvers and provides good facilities for integration with other libraries. Using these facilities I developed a module for PPM that can automatically export PPM domain decompositions as PETSc local-to-global mappings, PPM particle data as PETSc vectors and PSE operators as PETSc matrices. Both interfaces share underlying data so there is no copying involved and PPM and PETSc can be used interchangeably. With these utilities in place it is relatively easy to implement the equations in PETSc and use the provided solvers. As the module will be

![Figure 3.3: Convergence of the implicit scheme as a function of $dh$. The plot shows the final relative L2 error of the simulation. In each case $dt$ was chosen as the maximum time step that gives a final error comparable to Forward Euler.](image)
3.3. Implicit Time Stepping

included into PPM the interested reader can consult Appendix A on page 41 for more information.

For the time stepping scheme I used the second order Crank-Nicolson scheme

\[ u^{(t+1)} = u^{(t)} + dt \frac{1}{2} \left( a\Delta u^{(t+1)} + a\Delta u^{(t)} \right). \]  

(3.6)

Even though there is no stability limit on the size of the time step the accuracy of the solution imposes a limit because it goes down as \( dt \) grows. Furthermore as the solvers are also numeric they will require more iterations for bigger time steps. For excessively large \( dt \) the solvers might fail to converge. Thus \( dt \) should not be increased beyond the maximum of the ratio of \( dt \) to wall clock time, while also observing the restriction imposed by the required level of accuracy.

The relative l2 norm of the solution is shown in Figure 3.2 as a function of \( dt \). The smallest time step tested is the same as Forward Euler. The plot shows that the time step can indeed be increased by almost two orders of magnitude without impacting accuracy. The accuracy limit on \( dt \) can also be seen as the error starts growing rapidly once the limit is passed.

![Figure 3.4](image)

**Figure 3.4:** Maximum \( dt \) for which the final l2 norm is smaller than with explicit time stepping. The plot shows maximum \( dt \) as a function of \( dh \) for Crank-Nicolson and Forward Euler.
3. Implicit Solver

It is interesting to note that the time step limit also depends on the chosen spatial resolution and the accuracy of the two time stepping schemes. The reference simulation is done using the first-order accurate Forward Euler time stepping, while the chosen implicit scheme, Crank-Nicolson, is second order accurate. The difference in accuracy is more pronounced for smaller $dh$. Therefore, if the time step is chosen so that the accuracy is on the order of the reference simulation, the difference in time step size should also increase.

The plot in Figure 3.3 shows the relative $l_2$ norm of the solution as a function of $dt$ for both Crank-Nicolson and Forward Euler. The time step for the implicit scheme was chosen so that the final relative $l_2$ errors are very similar, as this plot demonstrates. The time step sizes of both schemes are shown in Figure 3.4 and we can see that the time step of the implicit scheme is indeed much larger. As the spatial resolution increases the time step of the second-order implicit scheme goes down as $dh$ rather than $dh^2$ imposed by the CFL condition of Forward Euler.

These results suggest that the cost of the implicit scheme can be amortized with the larger time step size and that the speedup will be greater for bigger spatial resolutions of the simulation. Figure 3.5 shows the speedup of the implicit solver as a function of $dh$ and we can see that this is indeed the case.
### 3.3. Implicit Time Stepping

![Graph showing speed up from using an implicit scheme](image)

**Figure 3.5:** The speed up from using an implicit scheme. The plot shows the total simulation time ratio of explicit to implicit scheme. Error bars are the 99.5% confidence interval.
Chapter 4

Adaptive resolution

The accuracy constraint on the time step size depends on the fastest dynamics in the simulation, and analogously the constraint on required resolution, or the total number of particles, is determined by the smallest phenomena that we want to resolve. If we use a fixed time step size and a uniform resolution of particles in space we have to make them small enough to satisfy the strongest constraint. As the dynamics of the simulated phenomena can vary during the course of the simulation and the length scales can vary in different regions of the simulation domain it makes sense to adapt the time step size and use a non-uniform distribution of particles that also adapts during the course of the simulation. Methods that try to do this are called adaptive resolution. Here I combine both time and space adaptation methods to get a fully adaptive simulation.

4.1 Adaptive Time Stepping

The adaptive time stepping scheme I will use is taken from [11]. The main idea of this scheme is to use a local approximation of the temporal error to guide the evolution of the time step size. If the error estimate is low the time step will be cautiously increased. If at some point the error estimate becomes too large the time step will be retried with a smaller $dt$.

Let $q$ be the ratio of the current and previous step sizes $q = \frac{dt_k}{dt_{k-1}}$. The local error estimate is given by

$$D_k = \frac{q}{1+q}||u^{k+1} - (1+q)u^k + qu^{k-1}||_\infty.$$  

This equation is an approximation of the second derivative of the field in time and for $q = 1$ it is the centered second difference formula. The infinity norm of the approximation is used as a measure of the local temporal error.
4. Adaptive resolution

The temporal evolution of the time step then depends on $D_k$ in the following way. Let $Tol$ be the maximum allowed temporal error and let the ratio $r$ be

$$r = \left( \frac{Tol}{D_k} \right)^{\frac{1}{p+1}},$$

where $p$ is the order of the time stepping scheme. This ratio, scaled down with a safety factor $\vartheta$ and limited by $r_{\text{min}} = 0.5$ and $r_{\text{max}} = 2$, is the growth multiplier for the time step size $dt$.

$$dt_{\text{new}} = \max(r_{\text{min}}, \min(r_{\text{max}}, \vartheta r)) dt_{\text{old}}.$$ 

If $D_k$ is greater than $Tol$ the time step will be rejected and calculated again with the smaller $dt$. The safety factor is there to ensure that the time step size will decrease without the need to reject and recompute time steps. Keeping $dt$ at $\vartheta = 0.8$ from the limit of the temporal error ensures that if $D_k$ starts growing and $r$ gets closer to 1 the time step will start decreasing on time.

**Figure 4.1:** Evolution of step size over time. The plot shows $dt$ as a function of time. The two horizontal lines are the fixed time step sizes of the explicit and implicit scheme for comparable final error.
4.1. Adaptive Time Stepping

Figure 4.2: Speedup of the adaptive time stepping scheme as a function of $dh$. The dashed line is the speedup of the reference simulation which is 1 at all $dh$.

If the rate of change of the solution varies rapidly during the course of the simulation $\vartheta$ can be decreased to give more room for adjustment.

Figure 4.1 shows the time step size as a function of time. We can see that the step size comes close to the accuracy limit of the implicit scheme but remains lower at first. As the solution settles down and the rate of change decreases, the time step is slowly increased to take advantage of the smaller temporal error.

The measured speedup, shown in Figure 4.2, is slightly lower than that of the implicit scheme with a fixed time step. This is to be expected as the adaptive scheme starts with a small $dt$ and increases it by at most doubling the value. The number of steps performed before the simulation reaches the adequate time step, and the fact that it stays below the fixed implicit time step size for a while, brings down the speedup of the simulation. This is amortized in longer simulations by the increase of $dt$ above the fixed value of the implicit scheme as the accuracy constraint starts permitting longer time steps.

To measure the speedup the simulations were run with a $Tol$ parameter that gives smaller final errors than the explicit scheme. As the resolution...
increases and the $dt$ of the explicit scheme is decreased, it takes more time steps to reach the final time. The tolerance also has to be decreased because when the simulation is run with a larger number of time steps the same tolerance for the temporal error adds up to a larger final error.

4.2 Self-Organizing Lagrangian Particles

The method used for spatial adaptivity is called Self-Organizing Lagrangian Particles [10] (SOP) and was developed in the MOSAIC group. This method is already implemented in PPM and will be available as a module in the next major release. The basic idea is for each particle to carry a pseudo force with a potential that will attract the particles to regions of space where more resolution is required and push them away from regions where they are not needed. This is done by using a potential similar to the one in Figure 4.3 and further scaling the strength of each particle to reflect the required resolution requirement.

![Figure 4.3: Example particle-particle interaction potential $y = \left(1 - \frac{1}{x}\right)^2$.](image)

The required resolution is given by the monitor function $\tilde{D}(x)$ which gives the smallest scale that should be resolved by the simulation at every point in space. This function should reflect the resolution requirements of the
solution field at current time and should therefore be a function of some property of the solution. There are many possible choices for this function so PPM allows any function to be passed to the SOP routines. Here I will use the following function

$$\tilde{D}(x) = \frac{D_0}{\sqrt{1 + (|\nabla u(x)| + \max(|\nabla^2 u(x)|, |\frac{\partial^2 u(x)}{\partial x \partial y}|))^2}}$$

This choice of the monitor function increases the inter-particle strength in regions where the combination of the norm of the gradient of the function, and the bigger of the two norms of second derivatives of the function, is large. This ensures that regions where the solution field is steepest get the most particles and that flat areas of the solution will have the least.

Figure 4.4: Example particle distribution plot after one run of SOP.
4. Adaptive resolution

The parameter $D_0$ sets the coarsest resolution in the computational domain or equivalently an upper bound on inter-particle spacing.

The value of this function is used to scale the interaction potential to the local resolution requirement. The interaction potential can also be any user defined function. The default is a function that is strongly repulsive when particles are too close, and increasingly attractive as the distance between particles increases, similar to the example in figure 4.3.

The sum of these pairwise potentials, the total potential energy, is then minimized by moving the particles. As this can be quite computationally expensive local search methods can be used to find local minima which give sufficiently good results. This prevents large displacements of the particles during the adaptation as it would take a very long time to move them using a local search method. To account for that particles are created and destroyed in under and over resolved regions respectively.

The result of one run of SOP can be seen in figure 4.4 where $D_0$ is set to the function given above.

The current run time of SOP prevents the repeated adaptation of the particles to the changing solution field. In the case of the Heat Equation the

![Figure 4.5: Speedup on multiple processors.](image)
solution doesn’t move but simply decays exponentially with time, so this allows the simulation to be run with only the initial adaptation performed.

4.3 Parallel Performance

The parallel performance tests were done using explicit and implicit time stepping for 1, 4, 16 and 64 processors. The speedup tests shown in Figure 4.5 were done using 640,000 particles in all cases while the scaleup tests in Figure 4.6 were done with 10,000, 40,000, 160,000 and 640,000 particles respectively.

We can see that both the speedup and scaleup for the implicit scheme were lower than that of the explicit scheme. This is expected because the implicit solver introduces more rounds of communication that will impact performance as the number of processors grows. The difference is negligible when the simulation is run on cores of one processor and goes up as soon as the number of processors increases.

Note that the speedup and scaleup of the implicit scheme shown here are in relation to the case of implicit scheme on one processor. Comparing to the explicit scheme on one processor the implicit scheme has much better results. The plot shown here is meant to examine the scalability of the implicit scheme itself and not the combined effect of parallel execution and implicit time stepping.
4. Adaptive resolution

Figure 4.6: Scaleup on multiple processors.
Chapter 5

Membrane Dynamics

5.1 Theoretical Background

The theory required to explain and understand the physics of cellular membranes and the phase field equation used in this simulation is vast and a comprehensive overview is out of the scope of this report. The phase field model used in this simulation was developed as a PhD thesis by Felix Campelo. For a detailed treatment of all the topics in this introduction please refer to [2]. Here I will only describe a minimal set of concepts that is required for further discussion.

5.1.1 Differential Geometry of Surfaces

There are two common ways to mathematically describe a surface in space. The first is called the parametric form and is defined by a vector field \( \vec{S} \) that maps a two dimensional coordinate system \( \Sigma \) onto a surface in space.

\[
\vec{S} : \Sigma \longrightarrow \mathbb{R}^3
\]

\[
\sigma = (\sigma_1, \sigma_2) \longrightarrow \vec{S}(\sigma) \in \mathbb{R}^3
\]

The other way is called the implicit form and is given by a set of points that satisfy a surface equation \( \mathcal{F}(x, y, z) = 0 \).

At any point \( x \) on a surface we can define the following properties. The unit normal \( \mathbf{n}(x) \) which is a vector perpendicular to the tangent plane of the surface at \( x \). The surface also has two principle radii of curvature \( R_1 \) and \( R_2 \). If we look at an infinitesimal surface element around \( x \) it has a constant curvature which we can examine in the two principal directions. If we interpret the line segments along these directions as arcs of a circle,
the principle radii of curvature would be the radii of the circles to which these arcs would belong. The inverses of the principle radii are called the principle curvatures $c_1$ and $c_2$ of the surface at $x$.

We can also define the total $J = c_1 + c_2$, mean $H = \frac{1}{2}(c_1 + c_2)$ and Gaussian curvature $G = c_1c_2$ which will be useful for defining the physical properties of membranes.

### 5.1.2 Physics of Cellular Membranes

Looking at the structure of a cellular membrane we see that the physics is very complex. Membranes are composed of a lipid bi-layer with many different types of molecules mixed in, attached to, or spanning the surface of the membrane. Some of these molecules modify the bending rigidity of the membrane, some induce local curvature in the membrane, and some serve as binding sites for the cytoskeleton which exerts additional force on the membrane.

The equation developed in the Campelo thesis only models the elastic forces of the membrane but it can be easily extended to include other phenomena. The choice was motivated by an observation on red blood cells that was made in the sixties. Looking at red blood cells deformed by an external force it was discovered that they relax to their characteristic biconcave shape within a fraction of a second after the external force is removed. This shows that the biconcave shape is a minimum energy state most likely caused by the bending energy of the membrane itself.

This motivated the formulation of the Canham-Helfrich model of the bending energy of a membrane

$$F = F_{SC} + F_G,$$  

where $F_{SC}$ is the energy due to the total curvature and $F_G$ is due to the Gaussian curvature. The total curvature energy is

$$F_{SC} = \frac{\kappa}{2} \int_{\Gamma} (J - c_0)^2 ds,$$  

where $\Gamma$ is the membrane surface, $J$ is the total curvature, and $\kappa$ is the bending modulus of the membrane. The parameter $c_0$ is the Spontaneous Curvature (SC) and allows for a non-zero preferred curvature of the membrane. The parameter need not be a constant and can also be a function of $x$. This allows the model to be extended with space dependent spontaneous curva-
5.2 Phase Field Method

Instead of tracking the position of a complex surface in three dimensions we will use an implicit definition of the surface. Phase field methods introduce a field in space that has two plateaus which for our application will be $-1$ and $1$ but can in principle be any values. The plateaus of the phase field represent two separate regions of space, the inside and the outside of the membrane. The value of the phase field changes abruptly but continuously from one value to the other in a narrow region of space that implicitly defines the position of the membrane.

To define the phase field for the membrane we will use the signed distance function $d(x)$ that gives us the minimal distance to the membrane for any point in space. This function is positive inside the membrane and negative outside. We define the phase field as

$$\phi(x) = \tanh \left( \frac{d(x)}{\sqrt{2}\epsilon} \right).$$

(5.4)

The tanh gives us the two plateaus and the parameter $\epsilon$ gives us control of the interface width. The signed distance function is also very useful because its first derivative is the unit normal of the membrane surface and its second derivative is the curvature tensor. Total and Gaussian curvature can be easily calculated from the curvature tensor so this provides a way to express the Canham-Hellfrisch bending energy in terms of the phase field $\phi$. In addition,
imposing the constraints that total surface area and enclosed volume must remain constant the following equation is derived (see chapter 5 of [2] for details):

$$\frac{\partial \phi}{\partial t} = \bar{\kappa} \nabla^2 \left\{ (3\phi^2 - 1 - 2\varepsilon C_0(x)\phi) \Phi_{\text{SC}}[\phi] - \varepsilon^2 \nabla^2 \Phi_{\text{SC}}[\phi] + \varepsilon^2 \tilde{\sigma}(x) \nabla^2 \phi \right\}.$$  (5.5)

The parameter $\bar{\kappa}$ is the Gaussian bending rigidity (saddle splay modulus) introduced above, $C_0(x) = \frac{c_0(x)}{\sqrt{2}}$ is the scaled, space dependent spontaneous curvature, $\tilde{\sigma}$ is a Lagrange multiplier controlling for constant area that can be interpreted as a surface tension, and $\Phi_{\text{SC}}$ is

$$\Phi_{\text{SC}} = -\phi + \phi^3 - \varepsilon^2 \nabla^2 \phi - \varepsilon C_0(1 - \phi^2).$$  (5.6)

In order to guarantee that the constraint on surface area remains valid throughout the complete evolution of the phase field the Lagrange multiplier

$$\sigma = \frac{\sqrt{2}}{6\varepsilon^3 \bar{\kappa}} \tilde{\sigma}$$  (5.7)

has to correct for errors by evolving according to the following equation

$$\sigma_{k+1}(x) = \sigma_k(x) + \alpha (a[\phi_k(x)] - a_0(x)),$$  (5.8)

where $\alpha > 0$ is the step size, $k$ is the discretised time, $a_0$ is the fixed local surface area, and $a[\phi]$ is the local surface area functional

$$a[\phi] = \frac{3}{4\sqrt{2}\varepsilon} (1 - \phi^2)^2.$$  (5.9)

### 5.3 Implementation

Like the Heat Equation, the simulation is implemented using PPM for the particle simulation features and PETSc for the implicit time stepping. Algorithm 5.1 shows the detailed initialization of the simulation, including the communication rounds, neighbor list computation, SOP calls and operator kernel precomputation. The main loop of the simulation is shown in Algorithm 5.2, again with communication rounds, SOP, adaptive time calls and all other details.
5.3. Implementation

Algorithm 5.1 Initialization of the simulation.
1: initialize MPI, PPM and PETSc
2: create the PPM topology
3: create the particles
4: do a global mapping
5: allocate and initialize particle properties
6: do a ghost mapping
7: create neighbor lists
8: if sop enabled then
9: run sop
10: do a ghost mapping
11: recompute neighbor lists
12: end if
13: compute Laplacian operator kernels

Algorithm 5.2 Main loop of the simulation.
1: while $t \leq t_{\text{end}}$ do
2: calculate $\phi_{t+dt}$
3: if use adaptive time then
4: calculate $D_k$ and $dt_{\text{new}}$
5: if $D_k > Tol$ then
6: $dt \leftarrow dt_{\text{new}}$
7: goto 2 {Step will be retried with a smaller $dt$.}
8: end if
9: end if
10: $t \leftarrow t + dt$
11: if use adaptive time then
12: $dt \leftarrow dt_{\text{new}}$
13: end if
14: if use sop then
15: do a ghost mapping
16: run sop
17: end if
18: do a ghost mapping
19: if use sop then
20: recompute neighbor lists
21: recompute Laplacian operator kernels
22: end if
23: end while
Calculation of the next step of the simulation can be done with any time stepping scheme. In the case of implicit schemes the PETSc non-linear solver has to be used as the governing equation is non-linear. The non-linear solver uses a line search technique to find the local minimum of the time evolution equation (5.10). The local minimum will also be global if the search is started from inside the basin of attraction of the global minimum. As the starting point is always fixed to the previous value of the field $\phi$ the choice of $dt$ will determine the position of and distance to the global minimum. We therefore expect to see a limit on $dt$ that will be imposed by the non-linearity of the time evolution equation.

$$F(\phi_{t+1}) = \phi_{t+1} - \phi_t - dt \frac{1}{2} (f(\phi_{t+1}) + f(\phi_t)) = 0$$ (5.10)

The non-linear solver uses the Jacobian of the time evolution function to determine the direction for the line search. PETSc can approximate the Jacobian using finite differences but this is computationally very expensive. Instead of this a function can also be passed to the solver that can compute the Jacobian for any given value of $\phi$. The computational cost of this calculation turned out to be a bottleneck in the simulation.

### 5.3.1 Jacobian Calculation

The Jacobian is a matrix of all the first order derivatives of a function that represents the best linear approximation of the function around a chosen point (Equation (5.11)).

$$J_F = \begin{bmatrix} \frac{\partial F_1}{\partial \phi_1} & \ldots & \frac{\partial F_1}{\partial \phi_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_m}{\partial \phi_1} & \ldots & \frac{\partial F_m}{\partial \phi_n} \end{bmatrix}$$ (5.11)

Here the derivatives of the time evolution function have the following structure

$$\frac{\partial F_i}{\partial \phi_j} = \frac{\partial \phi_i}{\partial \phi_j} - dt \frac{1}{2} \frac{\partial f_i(\phi)}{\partial \phi_j} = \delta_{ij} - dt \frac{1}{2} \frac{\partial f_i(\phi)}{\partial \phi_j},$$ (5.12)

where $f$ is the right hand side of the phase field equation (5.5). Evaluating the partial derivatives of $f$ requires substitution of the differential operators with the approximation which is used to compute them. In our case the approximation is DC-PSE:

$$\nabla^2 f_i = \sum_{k \in N(i)} \eta_{ik} (f_k - f_i),$$ (5.13)
where $\mathcal{N}(i)$ is the set of neighbors of particle $i$. First the outermost derivative is substituted:

$$\frac{\partial f_i(\phi)}{\partial \phi_j} = \bar{\kappa} \sum_{k \in \mathcal{N}(i)} (G_k - G_i), \quad (5.14)$$

where $G_p$ is the inside of the curly braces of the phase field equation (5.5). Proceeding with the substitution for all the Laplacians inside $G$ and applying the partial derivative with respect to $\phi_j$ we get:

$$G_k = (6 - 2\epsilon C_0)\delta_{kj} \Phi_{SC}[\phi]_k + (3\phi_k^2 - 1 - 2\epsilon C_0 \phi_k) \frac{\partial \Phi_{SC}[\phi]_k}{\partial \phi_j} \quad (5.15)$$

$$- \epsilon^2 \sum_{l \in \mathcal{N}(k)} \eta_{kl} \left( \frac{\partial \Phi_{SC}[\phi]_l}{\partial \phi_j} - \frac{\partial \Phi_{SC}[\phi]_k}{\partial \phi_j} \right) + \epsilon^2 c_k(\phi) \sum_{l \in \mathcal{N}(k)} \eta_{kl} (\delta_{lj} - \delta_{kj})$$

where the partial derivative of $\Phi_{SC}$ is:

$$\frac{\partial \Phi_{SC}[\phi]_l}{\partial \phi_j} = -\delta_{lj} + \delta_{lj} 3\phi_l^2 - \epsilon^2 \sum_{m \in \mathcal{N}(l)} (\delta_{mj} - \delta_{lj}) + \delta_{lj} 2\epsilon C_0 \phi_l. \quad (5.16)$$

Every term in this equation is multiplied by a $\delta_{pj}$ where $j$ is the index of a column of the Jacobian matrix. Looking at one row of the Jacobian, $i$, we can see that the non-zero structure depends on the $\delta_{pj}$ in the above equations. For any row $i$ the columns that are nonzero are the ones that are in the set of the three levels of neighbors of $i$. Separating the contribution of each level of neighbors by separating out the different $\delta_{pj}$ we can compute one whole row of the Jacobian by looping through the neighbors of $i$ and adding the contributions to the appropriate neighbor position. The third level contribution in the partial derivative of $\Phi_{SC}$ (5.16) does not depend on $\phi$ and can be precomputed. This precomputation is valid as long as the operator kernels are constant, which means that the particles must remain fixed in space. It also means that the precomputation has to be repeated every time SOP is run.

Algorithm 5.3 shows the precomputation of the third level neighbor contribution to the Jacobian. The cost is $O(N \cdot M^3)$ where $N$ is the number of particles and $M$ is the number of neighbors of a particle. The computation of the Jacobian, shown in Algorithm 5.4, is $O(N \cdot M^2)$.

If we fix a desired resolution for the simulation, or equivalently the number of particles per side of the domain $n$, the number of particles also grows from $n^2$ in 2D to $n^3$ in 3D. The typical values for $M$ are around 20 in 2D.
5. **Membrane Dynamics**

Algorithm 5.3 Jacobian precomputation.

1: allocate(ht(N))
2: count ← 0
3: for i = 1 → N do
4:     count ← count + 1
5:     insert(ht(i),i,count)
6:     index(i,count) ← i
7:     for all k ∈ neighbors(i) do
8:         if not search(ht(i),k) then
9:             count ← count + 1
10:            insert(ht(i),k,count)
11:            index(i,count) ← k
12:         end if
13:     for all l ∈ neighbors(k) do
14:         if not search(ht(i),l) then
15:             count ← count + 1
16:             insert(ht(i),l,count)
17:             index(i,count) ← l
18:         end if
19:     for all m ∈ neighbors(l) do
20:         if not search(ht(i),m) then
21:             count ← count + 1
22:             insert(ht(i),m,count)
23:             index(i,count) ← m
24:         end if
25:         precompute l_3(i,k,l,m)
26:     end for
27: end for
28: end for

and 50 in 3D. For \( n = 100 \) the cost of the precomputation is on the order of \( 10.000 \cdot 8.000 = 80.000.000 \) for 2D and \( 1.000.000 \cdot 125.000 = 125.000.000.000 \) for 3D. This method of computing the Jacobian fails in 3D as the precompute step takes roughly 15 hours to compute for this resolution. This is the reason why all of the simulations were done in 2D.

5.4 **Simulation**

The phase field is initialized by setting it to 1 inside a box \( x \in [0.2, 0.8], y \in [0.3, 0.7] \) and \(-1\) outside, on the domain \([0,1]^2\) with periodic boundary conditions (figure 5.1). As this initial condition does not have the tanh profile of
Algorithm 5.4 Jacobian computation.

1: precompute $\Phi_{SC}$, $m_1$ and $m_2$
2: $value = l_3$
3: for $i = 1 \rightarrow N$ do
4:     for all $k \in neighbors(i)$ do
5:         for all $l \in neighbors(k)$ do
6:             target = search($ht(i), l$)
7:             $value(i, target) = value(i, target) + l_2(i, l, k)$
8:         end for
9:     target = search($ht(i), k$)
10:    $value(i, target) = value(i, target) + l_1(i, l)$
11: end for
12: $value(i, 1) = value(i, 1) + l_0(i)$
13: $J(i, index(i)) = value(i)$
14: end for

the phase field, all simulations (explicit and implicit) are run with a certain number of explicit time steps to let the initial condition assume a smoother interface.

![Figure 5.1: Initial condition of the Phase Field simulation.](image)

The stability limit on time step size is $dt \approx 10^{-13}$ which results in a very slow simulation. Figure 5.2 shows the solution after four days and $21,460,000,000$
time steps on 16 processors. The membrane has not yet relaxed to a minimum energy state at this point and would likely take many days to do so.

![Image](image.png)

**Figure 5.2:** Solution of the Phase Field simulation with Forward Euler after four days and 21,460,000,000 steps on 16 processors.

### 5.4.1 Implicit Solver

Running the simulation with the implicit solver turned out to be slower than with Forward Euler. The gain in time step size was dwarfed by the cost of the non-linear solver and the computational cost per unit of simulated time was two to three orders of magnitude worse. Increasing the time step did not give any benefit as the non-linear solver requires proportionally more iterations. It is also not straightforward to find the maximum time step size that guarantees a small number of iterations of the non-linear solver. This is due to the fact that the number of iterations at each time step can vary significantly in the course of the simulation.

The problem is illustrated in Figure 5.3. As the phase field solution is changing very fast (due to the initial condition) I started the implicit solver at different points in simulated time. The different curves were produced by starting implicit time stepping after the number of Forward Euler steps shown in the legend. The plot shows the number of solver iterations per time step for the first twenty steps after implicit time stepping started. All the curves
were produced with the same $dt$.

![Graph showing number of SNES iterations per time step for different starting times of the implicit scheme.](image)

**Figure 5.3:** Number of SNES iterations per time step, for different starting times of the implicit scheme.

The number of iterations depends strongly on how fast the solution is moving. Starting implicit time stepping later gives a stable number of iterations per time step, while starting it too early gives very unstable behavior and the solver fails to converge in the extreme. This behavior was observed for other time step sizes as well. The benefit of running more explicit time steps decreases because the rate of change of the solution is also decreasing slower with time. Even when the explicit time stepping was run for two hours before starting the implicit solver the computational cost of the implicit solver was bigger than that of the explicit scheme.

This preliminary result indicates that the nonlinear system (5.5) is difficult to solve efficiently using the current straightforward approach, especially when the solution is stiff. Considerable improvement is to be expected, however, by operator splitting techniques and preconditioning, both of which are now readily available in the PPM framework.
5. Membrane Dynamics

5.4.2 Adaptive Time

The behavior of the adaptive time method is shown in figure 5.4. The plot shows the number of solver iterations on the left y axis, and the size of the time step on the right y axis, both as a function of the time step number. The solver starts from the time step size of Forward Euler and is increased slowly up to the limit imposed by the local temporal error. Once the limit is reached $dt$ is slowly increasing with the decreasing temporal error. This decrease in the temporal error is due to the slower rate of change of the solution. The number of solver iterations also grows as $dt$ is increased, reaches the maximum of 300 iterations and fails to converge. The same behavior was observed for smaller values of $Tol$ and for different starting times of the implicit solver.

![Figure 5.4](image)

*Figure 5.4:* Results of running the adaptive time method on the phase field equation. The plot shows $dt$ and the number of solver iterations as a function of the time step number.

The local temporal error limit on the time step size increases faster than the limit from the convergence of the non-linear solver. The time adaptivity method has to be extended to take the non-linearity of the solution into account and limit the number of non-linear solver iterations.
The goal of this master thesis was to develop a simulation of the phase field equation for membrane dynamics using adaptive resolution particle methods and to examine the performance of the simulation. During the course of the thesis an easy-to-use PETSc interface for PPM was developed and it was shown that PETSc can be effectively used for implicit time stepping in PPM particle simulations. The performance was tested on the heat equation where the results matched theoretical predictions well and gave a very good speedup compared to fully explicit time-stepping.

An adaptive-time method was implemented to dynamically vary the step size depending on the instantaneous stiffness of the solution. The method also provides an estimate of the local temporal error. Temporal adaptivity was then combined with spatial adaptation to give a fully adaptive method for particle simulations that was also tested and verified on the heat equation.

The first problem recognized is that the parallel speedup of the PPM library is excellent when run on processors with shared memory, but as soon as it is replaced with network communication the speedup goes down dramatically. This indicates that there is a problem with the way PPM communication is performed and that it should be examined in more detail.

The straightforward way of computing the Jacobian required by the implicit solver was good enough for the heat equation but it proved to be prohibitively expensive for the sixth-order phase field equation in 3D. Other ways of computing the Jacobian need to be examined for this method to be applicable.

The non-linear solver was shown to be very slow when solving the time evolution equation of the membrane simulation. One possible approach to addressing this is to try better preconditioners for the Jacobian matrix. Another would be the separation of the non-linear and linear parts of the
6. Conclusions

governing equation as the non-linear part would be of order 4 instead of 6. The non-linear part could be solved with an explicit method where the allowed time step for a fourth-order equation would be bigger, while the linear sixth-order part could be solved implicitly with a linear solver.

The adaptive time stepping scheme was shown to be inadequate for highly nonlinear equations. There, the time step size that optimizes the overall runtime for a given accuracy is not limited by the local temporal error but by the number of solver iterations required to compute one time step. Adding more constraints to the method that would take non-linearity and the number of solver iterations into account would likely solve this problem.

Due to time constraints spatial adaptivity still needs to be tested on the phase field equation. As the field is flat everywhere except for a small region around the surface of the membrane, the expected savings of computational resources are much greater than for the heat equation with the chosen initial condition.
The choice of PETSc for the implicit solver in the simulation was motivated by the fact that PETSc data structures are designed as an object oriented hierarchy that can be extended by the user. This flexibility in design made it possible to create a very simple interface between PPM and PETSc. To use the PETSc solvers in this thesis `ppm_module_petsc` was developed to create this interface.

PETSc already supports distributed vectors with ghost elements and they can be created on top of already allocated memory. The only thing required is a mapping between the local order of the elements in the vector and the global ordering. This allows for the PPM data structures to be wrapped into petsc vectors and operated on directly without the need for copying. To create this mapping each particle in PPM has to be assigned a unique global id. This can be done by calling:

```c
call compute_global_indices(info)
```

and should be done at initialization and whenever the number of particles changes. After the global indices have been computed, properties stored on the particles can be wrapped into PETSc vectors by calling:

```c
Vec x

x = petsc_vector(Particles, property_id)
```

DC operators defined in PPM can also be wrapped into PETSc matrices by calling:

```c
Mat A
```
A. PPM–PETSc utilities

\[ A = \text{petsc\_matrix}(\text{Particles}, \text{operator\_id}) \]

This also happens with no duplication of data. The vectors and matrices can then be used with all PETSc routines. For example, applying an operator to a vector can be done by calling:

\begin{verbatim}
integer err
Vec r

call MatMult(A, x, r, err)
\end{verbatim}

To create other PETSc matrices that can work with the vectors from PPM, the matrix needs to be passed a PETSc local to global mapping that is computed from PPM’s processor assignment. The mapping is computed with the global indices and can be accessed through the public variable \( lgm \).
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Convergence Plot for Forward Euler</td>
<td>12</td>
</tr>
<tr>
<td>3.2</td>
<td>Implicit Scheme Convergence for $dt$</td>
<td>13</td>
</tr>
<tr>
<td>3.3</td>
<td>Implicit Scheme Convergence for $dh$</td>
<td>14</td>
</tr>
<tr>
<td>3.4</td>
<td>Time Step Size</td>
<td>15</td>
</tr>
<tr>
<td>3.5</td>
<td>Speed Up of Implicit Scheme</td>
<td>17</td>
</tr>
<tr>
<td>4.1</td>
<td>Step Size Evolution</td>
<td>20</td>
</tr>
<tr>
<td>4.2</td>
<td>Adaptive Time stepping Speedup</td>
<td>21</td>
</tr>
<tr>
<td>4.3</td>
<td>SOP Interaction Potential</td>
<td>22</td>
</tr>
<tr>
<td>4.4</td>
<td>SOP Particle Distribution</td>
<td>23</td>
</tr>
<tr>
<td>4.5</td>
<td>Parallel Speedup</td>
<td>24</td>
</tr>
<tr>
<td>4.6</td>
<td>Parallel Scaleup</td>
<td>26</td>
</tr>
<tr>
<td>5.1</td>
<td>Initial Condition</td>
<td>35</td>
</tr>
<tr>
<td>5.2</td>
<td>Solution</td>
<td>36</td>
</tr>
<tr>
<td>5.3</td>
<td>SNES Iterations</td>
<td>37</td>
</tr>
<tr>
<td>5.4</td>
<td>Adaptive Time with Phase Field</td>
<td>38</td>
</tr>
</tbody>
</table>
List of Algorithms

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The general particle simulation algorithm.</td>
<td>9</td>
</tr>
<tr>
<td>5.1</td>
<td>Initialization of the simulation.</td>
<td>31</td>
</tr>
<tr>
<td>5.2</td>
<td>Main loop of the simulation.</td>
<td>31</td>
</tr>
<tr>
<td>5.3</td>
<td>Jacobian precomputation.</td>
<td>34</td>
</tr>
<tr>
<td>5.4</td>
<td>Jacobian computation.</td>
<td>35</td>
</tr>
</tbody>
</table>
Bibliography


